Bài tập lab 9

Bài 1:

- B1: Hàm **sum_of_numbers** được gọi với n = 7
- B2: Vì n không bằng 1, hàm sẽ chạy vào phần else và thực hiện **return 7 + sum_of_numbers(6)**
- B3: Tiếp theo, hàm **sum_of_numbers(6)** cần được tính toán. Nó sẽ thực hiện tương tự: Vì n (bây giờ là 6) không bằng 1, nó sẽ thực hiện **return 6 + sum_of_numbers(5)**
- B4: Quá trình này tiếp tục cho đến khi hàm **sum_of_numbers(1)** được gọi
- B5: Tại **sum_of_numbers(1)**, vì n bằng 1, hàm sẽ trả về 1 mà không cần gọi đệ quy
- B6: Sau đó, mỗi lần gọi hàm sẽ được giải quyết theo thứ tự ngược lại:
  + **sum_of_numbers(2)** trả về 2 + 1
  + Sau đó **sum_of_numbers(3)** trả về 3 + (2+1)
  + Cứ thế cho đến khi quay trở lại lần gọi ban đầu
  + Cuối cùng tính tổng tất cả các số từ 1 đến 7

⇨ Khi chạy chương trình, hàm này sẽ tính tổng tất cả các số từ 1 đến 7 và in ra tổng số khi được gọi

Bài 2:

- B1: Hàm fibonacci được gọi với n = 8
- B2: Vì n không nhỏ hơn hoặc bằng 1, hàm tiếp tục thực hiện trong khối **else**
- B3: Hàm gọi chính nó với n=7 (**fibonacci(7)**) và n=6 (**fibonacci(6)**)

- B4: Khi đạt đến các trường hợp cơ sở (khi n bằng 0 hoặc 1), hàm bắt đầu trả về các giá trị theo các lần gọi đệ quy
- B5: Các giá trị được trả về này được tổng hợp theo công thức đệ quy cho đến khi tất cả các lần gọi được giải quyết trở lại lần gọi ban đầu với n=8
- B6: Các giá trị được trả về này được tổng hợp theo công thức đệ quy cho đến khi tất cả các lần gọi được giải quyết trở lại lần gọi ban đầu với n=8

⇨ Cuối cùng, **fibonacci(7)** (trả về 13) và **fibonacci(6)** (trả về 8) được cộng lại để cho kết quả là 21, đây chính là số Fibonacci thứ 8

Bài 3:

- B1: **Gọi hàm ban đầu**: Ta bắt đầu bằng việc gọi hàm **power(2, 6)**
- B2: **Kiểm tra điều kiện**: Trong hàm power, ta kiểm tra nếu n = 0, thì trả vềNgược lại, ta thực hiện bước tiếp theo
- B3: **Bước đệ quy**: Vì n khác 0, ta thực hiện bước đệ quy: **return 2 * power(2, 5)**
- B4: **Tiếp tục đệ quy**: Bây giờ, ta gọi hàm **power(2, 5)**
- B5: **Lặp lại bước 3 và 4**: Ta tiếp tục lặp lại bước đệ quy cho đến khi n = 0
- B6: **Kết quả cuối cùng**: Khi n = 0, hàm trả về 1. Sau đó, ta truy ngược từ các bước đệ quy trước đó để tính tổng bằng 64

⇨ Vậy kết quả của 2^6 là 64

Bài 4:

- B1: Nếu số đĩa là 1, chuyển đĩa từ cọc A sang cọc B (Đây là điểm dừng của quá trình đệ quy)
- B2: Nếu số đĩa lớn hơn 1, thực hiện các bước sau:

+ 2.1: Chuyển **3 đĩa** từ cọc A sang cọc C, sử dụng cọc B làm trung gian

+ 2.2: Chuyển **đĩa cuối cùng** từ cọc A sang cọc B

+ 2.3: Chuyển **3 đĩa** từ cọc C sang cọc B, sử dụng cọc A làm trung gian

⇨ Quá trình đệ quy tiếp tục cho đến khi chúng ta đạt được trường hợp cơ bản. Khi đó, hàm sẽ dừng và in ra các bước chuyển đĩa theo quy tắc của bài toán tháp Hà Nội

Bài 5:

- B1: Hàm **cho_ga** nhận hai tham số: **tong_so_con** (Tổng số con) và **tong_so_chan** (Tổng số chân)
- B2: Đầu tiên, hàm kiểm tra nếu **tong_so_con** và **tong_so_chan** đều bằng 0, nếu đúng thì trả về 0 cho cả hai giá trị, có nghĩa là không có con nào cả
- B3: Tiếp theo, hàm kiểm tra nếu tổng số chân không chia hết cho 2, nếu đúng thì trả về -1 cho cả hai giá trị, có nghĩa là không có phương án hợp lệ nào
- B4: Hàm sử dụng một vòng lặp for để duyệt qua từng giá trị có thể của số chó từ 0 đến **tong_so_con**
- B5: Trong mỗi lần lặp, hàm tính số gà bằng cách lấy **tong_so_con** trừ đi cho
- B6: Hàm kiểm tra nếu tổng số chân của gà và chó **(ga * 2 + cho * 4)** bằng với **tong_so_chan** đã cho. Nếu đúng, hàm trả về số chó và số gà tương ứng
- B7: Nếu không tìm được phương án hợp lệ, hàm sẽ gọi đệ quy với **tong_so_con** giảm đi 1 và **tong_so_chan** giảm đi 4, tương ứng với việc loại bỏ một con chó và thử lại
- B8: Cuối cùng, nếu sau tất cả các lần gọi đệ quy mà vẫn không tìm được phương án hợp lệ, hàm trả về -1 cho cả hai giá trị, có nghĩa là không có cách nào phân biệt số gà và số chó dựa trên đề bài đã cho